Prof. Dr. Stefan Brass
Institut für Informatik
MLU Halle-Wittenberg

04.07.2019

Logische Programmierung & deduktive Datenbanken — Übungsblatt 10 —

Ihre Lösungen zu der Hausaufgabe e) schicken Sie bitte per EMail an den Dozenten (mit "`[lp19]`" in der Betreff-Zeile). Einsendeschluss ist Mittwoch, der 10. Juli. Aufgabe b) wird dort auch besprochen: Sie sollten vorher darüber nachdenken, müssen aber nichts abgeben.

Zum Selbststudium

a) Schauen Sie sich die folgenden Webseiten an. Sie brauchen für diese Aufgabe nichts abzugeben. Ziel ist, dass Sie einen Eindruck davon gewinnen, was es im WWW zum Thema dieser Vorlesung gibt, und dabei für sich nützliche Quellen entdecken.

- Hier eine Link-Liste zu Prolog vom Hohenstaufen-Gymnasium Kaiserslautern:

  [https://www.hsg-kl.de/faecher/inf/material/prolog/material/index.php]

- Die folgende Sammlung von Aufgaben zu Prolog stammt von Karsten Hönig, einem Informatik-Lehrer in Berlin:

  [http://www.hoenig.info/informatik/prolog/Word/]
  [InGN13%20-%20KI%20Aufgaben%20PROLOG%202.doc]

- Wir beschäftigen uns aktuell mit Benchmarks für Systeme der Logikprogrammierung (und auch relationale Datenbanken). OpenRuleBench ist eine Sammlung von Benchmarks für regelbasierte Systeme:

  [http://rulebench.semwebcentral.org/]

  Auf folgender Webseite haben wir unsere früheren Messergebnisse festgehalten:

  [http://users.informatik.uni-halle.de/˜brass/push/bench.html]

  Wir arbeiten gerade an einer neuen Version, die auch stärker relationale Datenbanken mit berücksichtigen wird.

- Viele moderne RDBMS-Systeme erlauben rekursive Sichtdefinitionen, so dass man z.B. die transitive Hülle berechnen kann. Der folgende Artikel ist eine Übersicht der Unterstützung rekursiver Sichten:

  [http://www-users.mat.umk.pl/˜eror/papers/dta-2010.pdf]

  Eine aktuelle Liste unterstützender Systeme findet sich in der Wikipedia:

  [https://en.wikipedia.org/wiki/Hierarchical_and_recursive_queries_in_SQL]

b) Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zu Kapitel 4 antworten?

- Welche Regeln sind "allowed"? Geben Sie zunächst die erste/einfachste Variante der Definition wieder. Warum ist diese Eigenschaft für die praktische Bottom-Up-Auswertung mit dem T_P-Operator wichtig? Geben Sie ein Beispiel für eine Regel, die nicht "allowed" ist, und erläutern Sie, welches Problem es bei der Berechnung des minimalen Modells geben würde.

  Hinweis: Beachten Sie aber, dass der Satz über den Zusammenhang des minimalen Modells mit dem kleinsten Fixpunkt des T_P-Operators ohne die Einschränkung auf "allowed"-Regeln gilt. Diese Eigenschaft ist nur für die praktische Berechnung durch Anwendung des Satzes wichtig.

- Wenn man auch eingebaute Prädikate im Regelrumpf erlaubt, z.B. "`<`", warum ist die einfache Definition zulässiger Regeln ("allowed", s.o.) nicht ausreichend? Was wäre eine einfache Korrektur? Warum ist diese Lösung doch sehr eingeschränkt? Würde es für einfache SQL-Anfragen ausreichen?

- Für die entgültige Definition zulässiger Regeln ("range-restricted rule given a binding pattern β for the head") haben wir angenommen, dass eine Menge erlaubter Bindungsmuster $valid(p)$ für jedes Prädikat p definiert sind. Was ist dann die Intuition der Definition einer bereichsbeschränkten Regel? Warum braucht man für die Bottom-Up-Auswertung mit dem T_P-Operator "stark bereichsbeschränkte" Regeln? Was ist da der Unterschied?

- Wie kann man Funktionssymbole von Prolog mit eingebauten Prädikaten in einer deduktiven Datenbank simulieren? Erläutern Sie das am Beispiel von Listen. Was wäre die wesentliche Einschränkung gegenüber Prolog? Wenn man in der deduktiven Datenbankbank auch eine funktionale Notation erlauben würde, was könnte man damit gegenüber Prolog gewinnen?

Denken Sie auch über folgende Fragen zu Kapitel 5 nach:

- Was ist das Symbol in Prolog für den Cut?
- Was genau macht der Cut?
- Geben Sie ein kurzes Beispiel für ein logisches Programm, an dem man die Wirkungsweise des Cut erläutern bzw. ausprobieren kann.
- Wozu ist der Cut gut? Erläutern Sie einige typische Anwendungen des Cut. Inwieweit kann er zur Verbesserung der Performance eines Prolog-Programms eingesetzt werden? Was ist eine typische Anwendung für Fallunterscheidungen?
- Warum ist der Cut problematisch?
- Geben Sie ein Beispiel für eine Prädikatdefinition, in der der Cut einen relativ subtilen Fehler verursacht (d.h. einen Fehler, den der Programmierer nicht oder nicht sofort bemerkt, und der deswegen besonders schwierig/gefährlich ist).

- Welche anderen Prolog-Operatoren bzw. Prädikate kann man oft anstelle des Cut verwenden?
- Erläutern Sie das "if-then-else" in Prolog.
- Welche Entsprechungen gibt es zwischen Datentypen in klassischen Programmiersprachen wie C, Pascal oder Java und Daten in Prolog? Welche Möglichkeiten gibt es, Arrays in Prolog zu repräsentieren?
- Ein wesentlicher Unterschied zwischen imperativen Sprachen und Prolog ist, dass man in imperativen Sprachen der gleichen Variable im Ablauf einer Berechnung mehrfach einen neuen Wert zuweisen kann, während das bei Prolog nur einmal möglich ist (sofern man nicht mit Backtracking zu einem früheren Berechnungszustand zurückkehrt). Welche Möglichkeiten gibt es, diese Einschränkung zu umgehen, wenn man einen imperativ formulierten Algorithmus nach Prolog übersetzen will? Welchen Vorteil hat die Lösung in Prolog?

Präsenzaufgaben

c) Schreiben Sie ein Prädikat `show_bindings`, das eine Datalog-Regel einliest (also ohne Funktionssymbole, Listen und andere strukturierte Terme), und für jedes Rumpfliteral Prädikat und Bindungsmuster ausgibt. Dabei ist das erste Vorkommen jeder Variable "free", jedes weitere Vorkommen "bound". Konstanten sind natürlich auch "bound" Argumente. Der Regelkopf ist für diese Aufgabe irrelevant. Wenn man z.B. die folgende Regel eingibt:

```
p(X,a) :- q(X, X, Y, c), r(Y, Z).
```

soll die folgende Ausgabe gedruckt werden:

```
q: fbfb
r: bf
```

Sie können die Regel mit `read(R)` einlesen (`read` kann beliebige Termstrukturen von Prolog parsen, auch mit Operatoren wie `:-` und "`,`"). Dann rufen Sie z.B. ein Prädikat `process_body(R)` auf. Dies könnte den Rumpf wie folgt abspalten (Fakten ohne Rumpf können bei dieser Aufgabe ignoriert werden):

```
process_rule(:-(_Head,Body)) :- process_body(Body).
```

Man könnte aus der mit "`,`" verknüpften Struktur der Rumpfliterale eine Liste erzeugen mit folgendem Prädikat:

```
body_list(Body,List) :-
    Body = ','(Lit,Rest) ->
        List = [Lit|RestList], body_list(Rest,RestList);
        List = [Body].
```


Bei der Verarbeitung dieser Liste könnten z.B. die in der Vorlesung besprochenen eingebauten Prädikate "`=..`" und `var` nützlich sein. Nachdem Sie für eine Variable die "free" Information ausgegeben haben, binden Sie sie an eine Konstante, so dass Sie dann bei jedem weiteren Vorkommen diese Konstante sehen.

d) Bei der einfachsten Variante des NIM-Spiels besteht der Spielzustand aus einer Anzahl n von Streichhölzern. Die beiden Spieler können abwechselnd 1 oder 2 Hölzer nehmen. Wer das letzte Holz nimmt, hat gewonnen. Es gibt viele weitere (auch komplexere) Varianten, wie der Eintrag in der Wikipedia erläutert:

[https://de.wikipedia.org/wiki/Nim-Spiel]

Schreiben Sie ein Prädikat, das mit einer gegebenen Anzahl n von Hölzern aufgerufen wird, und ausgibt, ob man den Gewinn erzwingen kann, und wenn ja, wieviel Hölzer (1 oder 2) man nehmen muss.

Sie könnten z.B. Prädikate `gewinn(`n`)` und `verlust(`n`)` definieren, wobei `gewinn(`n`)` wahr ist, wenn der Spieler, der an der Reihe ist, den Gewinn erzwingen kann. Entsprechend bedeutet `verlust(`n`)`, dass der Gegenspieler den Gewinn erzwingen kann. Nun gilt `gewinn(`n`)` sicher für $n = 1$ und $n = 2$. Außerdem gilt es, wenn mindestens einer der beiden erreichbaren Zustände $n-1$ und $n-2$ Verlust-Positionen sind. Entsprechend ist n eine Verlust-Position, wenn beide erreichbaren Spielzustände Gewinn-Positionen sind.

Alternativ können Sie ein Prädikat `bewertung(`n`,`b`)` definieren, wobei $b = +1$ Gewinn bedeutet (für den Spieler am Zug) und $b = -1$ Verlust (Unentschieden gibt es hier nicht). Wenn Sie am Zug sind, berechnen Sie das Maximum der Bewertungen für die erreichbaren Spielzustände. Wenn der Gegner am Zug ist, das Minimum. Deswegen heißt dieses Verfahren der MinMax-Algorithmus:

[https://de.wikipedia.org/wiki/Minimax-Algorithmus]

Wenn der Spielbaum zu groß ist, kann man die Suche auch vorzeitig abbrechen, und muss den letzten Zustand dann mit einer Heuristik bewerten. In dieser Situation sind auch andere Zahlen außer -1, ggf. 0 und $+1$ möglich.

Hausaufgabe

Die folgende Aufgabe setzt das "Vier gewinnt" Spiel vom letzten Aufgabenblatt fort.

e) Es ist ein Turnier der "Vier gewinnt" Programme geplant. Sie können die Hausaufgabe in dieser Woche nutzen, Ihr Programm nochmals zu verbessern. Die einzige Forderung ist, dass Sie nicht das gleiche Programm nochmals abgeben — es sei denn, Sie haben letzte Woche schon sehr viel Zeit in die Spielstrategie investiert. Dann reicht eine EMail, dass Sie Ihr Programm unverändert lassen wollen. Sie sollten dann aber in wenigen Zeilen Ihren Algorithmus skizzieren. Andernfalls beschreiben Sie bitte kurz die Änderung.